COMPUTER SOCIETY OF INDIA (CSI)
COEP TECH STUDENT CHAPTER

PRESENTS

CODE QUEST 5.0

EDITIORIAL

KUNAL'S DILEMMA

EXPLANATION

The problem requires Kunal to allocate his available time fractionally across multiple slots to meet the required work quotas for both ASCI and CSI on a given day. The goal is to determine whether he can successfully distribute his time to satisfy the daily requirements.

1. Sorting Slots by Efficiency:
   - Each time slot has two productivity values: (A_i) (ASCI work) and (C_i) (CSI work).
   - To optimize time allocation, we compute the efficiency ratio (Ai/Ci) (if (C_i != 0), otherwise consider it as infinitely large).
   - We then sort the slots in descending order of this efficiency ratio, prioritizing slots where ASCI work is more efficient.

EXPLANATION

2. Prefix Sums for Efficient Lookups:
   - We maintain a prefix sum array (C[]), where (C[i]) stores the cumulative ASCI work if all time from the first (i) slots is dedicated to ASCI.
   - We also maintain a suffix sum array (E[]), where (E[i]) stores the cumulative CSI work if all remaining time from slot (i) onward is dedicated to CSI.

3. Checking Daily Requirements Efficiently:
   - For each day, given required ASCI work (A_total) and CSI work (C_total):
     - We find the smallest prefix index (i) where (C[i] >= A_total). This tells us how many slots are needed to satisfy the ASCI work requirement.

EXPLANATION

- We then compute how much time fractionally remains for CSI work.
    - If the total achievable CSI work (from partial and remaining slots) is enough, the answer for that day is 'P' (Possible), otherwise 'N' (Not possible).

4. Efficiency Considerations:
   - Sorting the slots takes (O(S log S)).
   - Building prefix and suffix sums takes (O(S)).
   - Checking for each day's requirement is done in (O(log S)) using binary search.
   - This ensures the approach efficiently handles large constraints.

EXAMPLE

Input:
1
1 3
3 2
4 1
2 3
5 5

Explanation of input:
- 1 test case
- 1 day and 3 slots
- Slots:
  - Slot 1: (3 ASCI, 2 CSI)
  - Slot 2: (4 ASCI, 1 CSI)
  - Slot 3: (2 ASCI, 3 CSI)
- Required work on Day 1: 5 ASCI, 5 CSI

EXAMPLE

Step 1: Compute Efficiency and Sort
- Compute efficiency ratios:
  - Slot 1: (3/2 = 1.5)
  - Slot 2: (4/1 = 4.0) (Highest, prioritize ASCI)
  - Slot 3: (2/3 approx 0.67) (Least efficient for ASCI)
- Sort in decreasing order:
  - Slot 2: (4 ASCI, 1 CSI)
  - Slot 1: (3 ASCI, 2 CSI)
  - Slot 3: (2 ASCI, 3 CSI)

Step 2: Compute Prefix and Suffix Arrays
- Prefix sum for ASCI:
  - C[0] = 4
  - C[1] = 4 + 3 = 7
  - C[2] = 7 + 2 = 9
- Suffix sum for CSI:
  - E[2] = 3
  - E[1] = 2 + 3 = 5
  - E[0] = 1 + 5 = 6

EXAMPLE

Step 3: Check Requirements
- Need 5 ASCI and 5 CSI
- Find ASCI requirement:
  - (C[0] = 4) (insufficient)
  - (C[1] = 7) (sufficient, so need up to slot 1)
- Compute fractional usage:
  - Need extra 1 ASCI from slot 2.
  - Slot 2: (1/3) of time spent on ASCI Leaves (2/3) for CSI.
  - (2/3 times 2 = 1.33) CSI from slot 2.
  - Total CSI: (1.33 + 5 = 6.33) (Enough!)
- Result: 'P'

Final Output:

P

SOLUTION

```python
import bisect
def solve():
    # Read number of days and slots
    num_days, num_slots = [int(_) for _ in input().split()]
    daily_requirements = []
    slots = []
    # Read slot values
    for _ in range(num_slots):
        slots.append([int(_) for _ in input().split()])
    # Read daily requirements
    for _ in range(num_days):
        daily_requirements.append([int(_) for _ in input().split()])
    # Sort slots based on efficiency (ASCI work per unit CSI work)
    slots.sort(key=lambda x: float(x[0]) / x[1] if x[1] else 100000)
    # Compute prefix sum for ASCI work
    asci_prefix_sum = []
    total_asci = 0
    for slot in slots:
        total_asci += slot[1]
        asci_prefix_sum.append(total_asci)
    # Compute suffix sum for CSI work
    csi_suffix_sum = []
    total_csi = 0
    for slot in reversed(slots):
        total_csi += slot[0]
        csi_suffix_sum.append(total_csi)
    csi_suffix_sum = csi_suffix_sum[::-1]
    result_string = ""
    # Process each day's requirements
    for required_csi, required_asci in daily_requirements:
        slot_index = bisect.bisect_left(asci_prefix_sum, required_asci)  # Find minimum slot index to meet ASCI requirement
        if slot_index == num_slots:  # If requirement can't be met, add 'N'
            result_string += "N"
            continue
        remaining_asci = asci_prefix_sum[slot_index] - required_asci  # Remaining ASCI work
        # Convert excess ASCI work to CSI work
        converted_csi = float(remaining_asci * slots[slot_index][0]) / slots[slot_index][1] if slots[slot_index][1] else slots[slot_index][0]
        # Total possible CSI work
        total_possible_csi = converted_csi + csi_suffix_sum[slot_index+1] if slot_index+1 < num_slots else converted_csi
        if total_possible_csi >= required_csi:  # Check if CSI requirement is met
            result_string += "P"
        else:
            result_string += "N"

    return result_string

num_test_cases = int(input())
for _ in range(num_test_cases):
    result = solve()
    print(result)

```